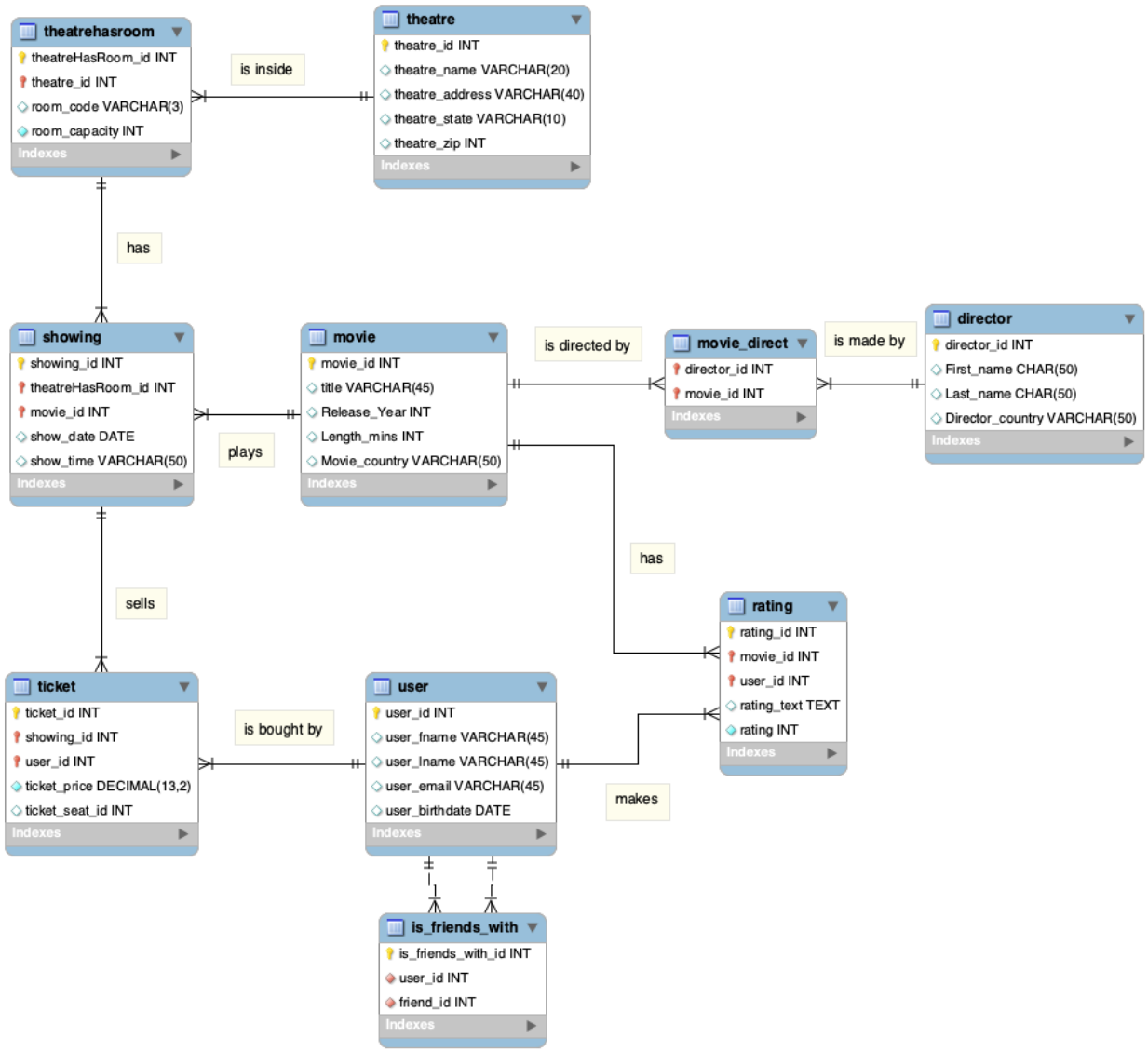
Milestone 2 – Generate SQL Queries for Database:
Movie Theater Chain Ticketing and Social Media Platform

Team 2-b:
Jack Bentsen
PO Yu Chao
Chia Yu Hsu
Zhou Jin

Internal Model:



Query 1:
This query helps user to find out the movie(s) directed by Joss Whedon. Clients can use similar queries to find out movies directed by their preferred/ interested directors.

```
SELECT movie.title, director.first_name, director.last_name
FROM movie_direct
JOIN movie USING (movie_id)
JOIN director USING (director_id)
WHERE director.first_name = 'Joss' AND director.last_name = 'Whedon';
```

Output:

	title	first_name	last_name
▶	The Avengers	Joss	Whedon
	Avengers 2 Age of Ultron	Joss	Whedon

Query 2:
This query helps user to find all the movies with more than one director. This data provides some insights on the directors' information if clients are interested.

```
SELECT movie.movie_id, movie.title, count(director.director_id) as `director number`
FROM movie_direct
JOIN movie USING (movie_id)
JOIN director USING (director_id)
GROUP BY movie.movie_id
HAVING `director number` > 1;
```

Output:

movie_id	title	director numb...
1	Avengers Endgame	2
6	Avengers Infinity War	2
17	Captain Marvel	2

Query 3:
This query helps user to find out the top three longest movies in terms of time (unit: minutes). This data can help clients to find movies to fit with their schedule depending on how much time they'd be able to spend on watching.

```
SELECT movie_id, title, length_mins as 'Runtime'
FROM movie
ORDER BY length_mins desc
limit 3;
```

Output:

movie_id	title	Runtime
1	Avengers Endgame	181
9	Transformers 4 Rebirth from Extinction	165
4	Transformers 3	154

Query 4:
This query displays the movies that were played in the VIP rooms in theatre during year 2019. Clients may use similar queries to find out showing of certain movies that would be available for VIP rooms if they'd like to have VIP rooms during ticket booking process.

```
SELECT movie.movie_id, movie.title, showing.show_date, theatreHasRoom.room_code
FROM movie
NATURAL JOIN showing
JOIN theatreHasRoom ON showing.theatreHasRoom_id = theatreHasRoom.theatreHasRoom_id
WHERE theatreHasRoom.room_code = 'VIP' AND showing.show_date LIKE '2019%'
ORDER BY showing.show_date;
```

Output:

movie_id	title	show_date	room_code
19	Star Wars: the rise of skywalker	2019-03-03	VIP
19	Star Wars: the rise of skywalker	2019-04-19	VIP
1	Avengers Endgame	2019-07-17	VIP
13	Spider Man : Far From Home	2019-11-25	VIP

Query 5:
This query displays movies whose showing dates are between 2019/11/15 and 2019/11/21. Also, it displays in which theatre the movies are shown. Similar queries could provide information of movies' showing date, title, length and the related theatre information by simply selecting different show dates range.

```
SELECT showing.show_date, movie.title, movie.Length_mins,
theatre.theatre_name, theatre.theatre_address, theatre_state
FROM showing
INNER JOIN movie ON showing.movie_id=movie.movie_id
NATURAL JOIN theatrehasroom JOIN theatre ON theatrehasroom.theatre_id=theatre.theatre_id
WHERE showing.show_date BETWEEN '2019-11-15' AND '2019-11-21'
ORDER BY show_date ASC;
```

Output:

show_date	title	Length_mins	theatre_name	theatre_address	theatre_state
2019-11-16	Toy Story	100	Amc Theaters	36 Maryland Crossing	WA
2019-11-17	Joker	122	AMC Oak Tree 6	22 Fair Oaks Crossing	WA
2019-11-20	Captain Marvel	124	The Grandview	50 Gerald Crossing	WA

Query 6:
This query shows the number of tickets sold and the total values of each showing sorted by the order of the descending total sales amount and showing id. This data provides us some insights on the selling market of the movies selected; also, it shows us the popularity of the movies by their total sales amount order.

```
SELECT showing.showing_id AS `ID`, showing.show_date AS `Show Date`, showing.show_time AS `Show Time`,
movie.title AS `Movie Title`,
COUNT(ticket_id) AS `Number of Tickets Sold`,
SUM(ticket_price) AS `Total Sales`
FROM ticket
JOIN showing USING (showing_id)
JOIN movie USING (movie_id)
GROUP BY showing.showing_id
ORDER BY `Total Sales` DESC, showing.showing_id;
```

Output:

ID	Show Date	Show Time	Movie Title	Number of Tickets Sold	Total Sales
23	2019-08-18	20:54:00	Captain Marvel	8	123.95
1	2019-10-22	19:08:00	Spider Man : Far From Home	7	114.66
19	2019-04-19	16:45:00	Toy Story	6	102.39
44	2017-06-23	23:48:00	Beauty and the Beast	7	100.24
28	2018-03-03	14:32:00	Demon Slayer Infinte Train	7	97.69
20	2019-06-23	10:18:00	Avengers Endgame	6	89.43
34	2019-10-21	23:28:00	Avengers Endgame	6	89.23
43	2018-01-19	23:09:00	Avengers Infinity War	6	88.14
15	2019-02-09	18:41:00	Spider Man : Far From Home	7	87.97
32	2019-03-19	12:35:00	Star Wars: the rise of skywalker	6	87.93
38	2018-05-22	21:25:00	Black Panther	6	83.39
18	2019-04-19	16:57:00	Star Wars: the rise of skywalker	6	81.53
36	2017-04-19	12:44:00	Beauty and the Beast	5	81.14
50	2017-04-09	16:25:00	Beauty and the Beast	5	80.99
46	2017-04-09	13:44:00	Beauty and the Beast	5	79.16
16	2019-07-21	20:12:00	Captain Marvel	5	77.11
14	2019-03-03	12:44:00	Demon Slayer Infinte Train	5	76.89
45	2017-04-10	20:57:00	Furious8	5	75.39
41	2017-08-16	19:43:00	Furious8	5	73.22
35	2018-02-10	22:09:00	Black Panther	5	69.11
21	2019-09-18	20:05:00	Demon Slayer Infinte Train	5	68.45

Query 7:
This query shows the available seats for each movie showing. Clients can use similar queries to find out seats available during the ticket reservation/ booking process.

```
SELECT showing.showing_id AS `Showing ID`, showing.show_date AS `Date`, showing.show_time AS `Time`,
ABS(theatrehasroom.room_capacity - COUNT(ticket.ticket_id)) AS `Available Seats`
FROM showing
JOIN theatrehasroom USING (theatrehasroom_id)
NATURAL JOIN ticket
GROUP BY showing.showing_id
ORDER BY `Available Seats`, showing.showing_id;
```

Output:

	Showing ID	Date	Time	Available Seats
▶	45	2017-04-10	20:57:00	70
	18	2019-04-19	16:57:00	74
	25	2019-03-03	18:24:00	76
	8	2019-11-25	17:22:00	77
	49	2018-10-31	13:09:00	79
	19	2019-04-19	16:45:00	84
	20	2019-06-23	10:18:00	84
	1	2019-10-22	19:08:00	93
	28	2018-03-03	14:32:00	93
	36	2017-04-19	12:44:00	95
	41	2017-08-16	19:43:00	95
	9	2019-07-10	11:05:00	97
	10	2019-09-18	12:29:00	97
	27	2019-07-17	18:26:00	97
	48	2017-09-19	16:32:00	97
	40	2017-08-16	20:08:00	98
	13	2019-07-17	14:06:00	111

Query 8:
This query shows the highest rated movies from everyone. Clients can use this query to find movies with highest average ratings.

```
SELECT movie.title as `Title`, ROUND(AVG(rating.rating),2) as `Average Rating`
FROM movie
NATURAL JOIN rating
GROUP BY movie.movie_id
ORDER BY `Average Rating` DESC;
```

Output:

	Title	Average Rating
▶	Aquaman	4.00
	Star Wars: the rise of skywalker	4.00
	Transformers 3	3.40
	Furious8	3.38
	Avengers Endgame	3.33
	The Avengers	3.20
	Captain Marvel	3.17
	Furious 7	3.00
	Transformers 4 Rebirth from Extinction	3.00
	Avengers 2 Age of Ultron	3.00
	Black Panther	2.67
	Joker	2.67
	Jurassic World Fallen Kingdom	2.60
	Spider Man : Far From Home	2.50
	Jurassic World	2.20
	Toy Story	2.17
	Avengers Infinity War	2.00
	Beauty and the Beast	2.00
	Demon Slayer Infinte Train	1.60
	Iron Man 3	1.25

Query 9:
This query shows movies with the largest number of ratings. Clients may use this query to search for movies with the most ratings and see if the movies are popular based on the number of ratings that the movies get.

```
SELECT movie.movie_id as `ID`, movie.title as `Title`, COUNT(rating_id) as `Number of Ratings`
FROM movie
NATURAL JOIN rating
GROUP BY movie.movie_id
ORDER BY `Number of Ratings` DESC;
```

Output:

ID	Title	Number of Ratings
3	Furious 7	9
5	Furious8	8
13	Spider Man : Far From Home	8
6	Avengers Infinity War	6
9	Transformers 4 Rebirth from Extinction	6
17	Captain Marvel	6
20	Toy Story	6
2	Jurassic World	5
4	Transformers 3	5
7	Demon Slayer Infinte Train	5
8	Jurassic World Fallen Kingdom	5
11	The Avengers	5
10	Iron Man 3	4
12	Avengers 2 Age of Ultron	4
15	Beauty and the Beast	4
16	Aquaman	4
1	Avengers Endgame	3
14	Black Panther	3
18	Joker	3
19	Star Wars: the rise of skywalker	1

Query 10:
This query shows the total sales for each movie showing and is grouped together by its movie id, showing date and showing time. It provides statistics on the movies sales and showing counts for the movie.

```
SELECT showing.show_date as `showing date`, showing.show_time as `showing time`,
movie.movie_id as `movie id`, movie.title as `movie title`,
SUM(ticket.ticket_price) AS `total sales`
FROM ticket
INNER JOIN showing
ON ticket.showing_id = showing.showing_id
INNER JOIN movie
ON movie.movie_id = showing.movie_id
GROUP BY showing.show_date, showing.show_time, movie.movie_id;
```

Output:

showing date	showing time	movie id	movie title	total sales
2019-07-16	14:24:00	1	Avengers Endgame	62.27
2019-07-17	14:06:00	1	Avengers Endgame	61.96
2019-06-23	10:18:00	1	Avengers Endgame	89.43
2019-07-17	18:26:00	1	Avengers Endgame	44.87
2019-10-21	23:28:00	1	Avengers Endgame	89.23
2017-08-16	20:08:00	5	Furious8	31.88
2017-08-16	19:43:00	5	Furious8	73.22
2017-07-05	12:22:00	5	Furious8	42.22
2017-04-10	20:57:00	5	Furious8	75.39
2017-09-19	16:32:00	5	Furious8	47.07
2018-01-19	23:09:00	6	Avengers Infinity…	88.14
2018-10-31	13:09:00	6	Avengers Infinity…	13.79
2019-06-02	16:56:00	7	Demon Slayer Infi…	48.92
2019-03-03	12:44:00	7	Demon Slayer Infi…	76.89
2019-09-18	20:05:00	7	Demon Slayer Infi…	68.45

Query 11:
This query finds out the director for the Avengers movie series and shows the average ratings for each one in the series. Users can use similar queries to find the movie/ movie series director information and the average ratings on the movie/ movie series selected.

```
SELECT movie.title, movie.Length_mins, movie.Release_Year,
ROUND(AVG(rating.rating),2) As `average rating`,
director.First_name, director.Last_name
FROM Movie_direct
JOIN movie USING (movie_id)
JOIN director USING (director_id)
NATURAL JOIN rating
GROUP BY movie.title
HAVING movie.title LIKE '%avengers%'
ORDER BY movie.Release_Year DESC;
```

Output:

	title	Length_mins	Release_Year	average rating	First_name	Last_name
▶	Avengers Endgame	181	2019	3.22	Anthony	Russo
	Avengers Infinity War	149	2018	3.05	Anthony	Russo
	Avengers 2 Age of Ultron	141	2015	2.90	Joss	Whedon
	The Avengers	143	2012	3.26	Joss	Whedon

Stored Procedure 1:
This stored procedure helps user to find all his/ her friends in the database. Users will be able to quickly find his/ her friends list by calling this stored procedure.

```
DROP PROCEDURE IF EXISTS getAllUserFriends;
DELIMITER $$
CREATE PROCEDURE getAllUserFriends(Puser_id INT)
BEGIN
    SELECT user.user_fname AS `First Name`, user.user_lname AS `Last Name`, user.user_email AS `Email`,
    user.user_birthdate AS `DOB`
    FROM is_friends_with
    JOIN user ON is_friends_with.friend_id = user.user_id
    WHERE is_friends_with.user_id = Puser_id;
END $$
DELIMITER ;
```

Example call:

```
CALL getAllUserFriends(5);
```

Output from example call:

First Name	Last Name	Email	DOB
Rowe	Bazelle	rbazelle3h@army.mil	1975-05-30
Lothaire	Shuker	lshuker9@geocities.jp	1966-05-16
Ardath	Bowry	abowryo@whitehouse.gov	2001-06-24
Elsy	Fetherstone	efetherstone2q@moonfruit.com	1991-10-20
Reginald	Whitcombe	rwhitcombe11@webmd.com	1945-06-08
Guinna	Mawd	gmawd2o@cocolog-nifty.com	1930-08-03
Raphaela	Goater	rgoater4d@ehow.com	1949-01-09

Stored procedure 2:
This stored procedure gets all the ratings from the given user. User can use this stored procedure to get all his/her ratings on movies as a sorted table.

```
DROP PROCEDURE IF EXISTS getAllUserRatings;
DELIMITER //
CREATE PROCEDURE getAllUserRatings(Puser_id INT)
BEGIN
    SELECT rating.user_id, movie.title, rating.rating_text, rating.rating
    FROM rating
    JOIN movie USING (movie_id)
    WHERE user_id = Puser_id
    ORDER BY rating.rating DESC;
END //
DELIMITER ;
```

Example call:

```
CALL getAllUserRatings(1);
```

Output from example call:

user_id	title	rating_text	rating
1	Spider Man : Far From Home	Nam ultrices, libero non mattis pulvinar, nulla pe…	4
1	Transformers 4 Rebirth from Extinction	In congue. Etiam justo. Etiam pretium iaculis justo.	1
1	Avengers Infinity War	Integer ac leo. Pellentesque ultrices mattis odio.…	1

Stored procedure 3:
This stored procedure gets the highest average rating on movies they have rated, and this would help friends to make recommendations easier by checking their highest ratings on movies.

```
DROP PROCEDURE IF EXISTS getHighestRatedMoviesAmongFriends;
DELIMITER !!
CREATE PROCEDURE getHighestRatedMoviesAmongFriends(Puser_id INT)
BEGIN
    SELECT movie.title AS `Title`, AVG(rating.rating) AS `Average Rating Among Friends`
    FROM rating
    JOIN movie USING (movie_id)
    WHERE rating.user_id IN (
        SELECT friend_id
        FROM is_friends_with
        WHERE user_id = 5
    )
    GROUP BY movie.title
    ORDER BY `Average Rating Among Friends` DESC;
END !!
DELIMITER ;
```

Example call:

```
CALL getHighestRatedMoviesAmongFriends(45);
```

Output from example call:

Title	Average Rating Among Friends
Joker	3.5000
Beauty and the Beast	3.5000
Jurassic World	3.0000
Captain Marvel	3.0000
Furious8	3.0000
Black Panther	2.0000
Avengers Infinity War	2.0000
Toy Story	2.0000
Transformers 4 Rebirth from Extinction	2.0000
Demon Slayer Infinte Train	2.0000
Transformers 3	1.0000
Jurassic World Fallen Kingdom	1.0000